Classification Models and Hyperparameter Finetuning

Data

	home_Goals	away_Goals	home_GoalsHalfTime	home_xGoals	home_shots	home_ppda	h
5437	2	2	2	1.367870	9.0	31.6000	
5438	3	3	1	1.396890	14.0	5.7429	
5439	0	3	0	0.813737	9.0	7.0000	
5440	0	2	0	0.632940	6.0	16.0625	
5441	0	0	0	1.544680	14.0	3.6087	
...	...	...	...	...	...	...	
12675	1	2	1	1.411190	15.0	12.3684	
12676	1	2	1	1.198190	10.0	16.2632	
12677	2	0	1	1.332690	12.0	8.2857	
12678	0	1	0	1.460500	19.0	7.5600	
12679	1	1	1	0.323960	6.0	15.1000	

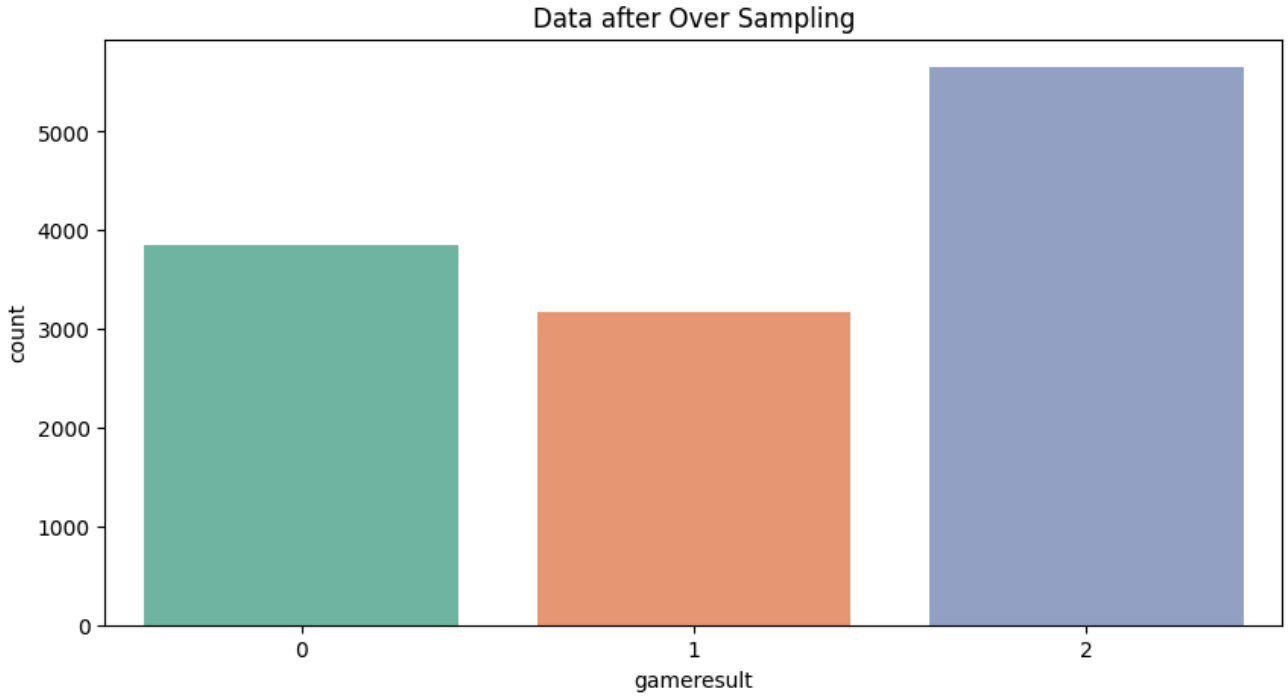
12680 rows × 19 columns

```
(12680, 19)
```

```
(2536, 20)
```

Visual of the data created on stage 5 with RandomOverSample techniqe

```
[Text(0.5, 1.0, 'Data after Over Sampling')]
```




```
2    5654
0    3854
1    3172
Name: gameresult, dtype: int64


10682    0
10794    1
2455     2
6463     0
168      1
        ..
3475     1
1487     1
4070     0
4404     2
9411     2
Name: gameresult, Length: 2536, dtype: int64
```

Supervised Models

```
home_Goals                  int64
away_Goals                  int64
home_GoalsHalfTime          int64
home_xGoals               float64
home_shots                float64
home_ppda                 float64
home_corners              float64
home_total_assists          int64
away_xGoals               float64
away_ppda                 float64
away_total_assists          int64
away_total_red_cards        int64
home_shotsOnTarget_cat    float64
away_shotsOnTarget_cat    float64
home_total_assists_cat    float64
away_total_assists_cat    float64
home_Goals_cat            float64
away_Goals_cat            float64
split                       int8
dtype: object
```

Linear Regression

```
Series([], dtype: float64)
```

```
home_Goals                  int64
away_Goals                  int64
home_GoalsHalfTime          int64
home_total_assists          int64
away_total_red_cards        int64
away_total_assists          int64
home_Goals_cat            float64
away_total_assists_cat    float64
home_total_assists_cat    float64
away_shotsOnTarget_cat    float64
home_shotsOnTarget_cat    float64
away_ppda                 float64
away_xGoals               float64
home_corners              float64
home_ppda                 float64
home_shots                float64
home_xGoals               float64
away_Goals_cat            float64
split                     float64
dtype: object
```

col_0	0	1	2
gameresult			
0	754	0	0
1	0	639	0
2	0	0	1143

Decision Tree

col_0	0	1	2
gameresult			
0	754	0	0
1	0	634	5
2	0	0	1143

Random Forest

col_0	0	1	2
gameresult			
0	754	0	0
1	1	634	4
2	0	0	1143

Adaptive Boosting (ADABoost)

col_0	0	1	2
gameresult			
0	409	345	0
1	0	639	0
2	0	512	631

Gradient Boosting Machine (GBM)

col_0	0	1	2
gameresult			
0	754	0	0
1	2	637	0
2	0	0	1143

Support Vector Machine (SVM)

col_0	0	1	2
gameresult			
0	748	5	1
1	3	635	1
2	0	4	1139

XGBoost parallel tree boosting (GBDT, GBM)

col_0	0	1	2
gameresult			
0	754	0	0
1	2	634	3
2	0	0	1143

Model Selection

	model	Accuracy	Precision	Recall	f1-score	Log-loss	AUC
0	Logistic Regression	1.000000	1.000000	1.000000	1.000000	0.005162	1.000000
6	XGB	0.998028	0.998034	0.998028	0.998026	0.005235	1.000000
2	RandomForest	0.998028	0.998034	0.998028	0.998026	0.014983	0.999991
4	GBM	0.999211	0.999213	0.999211	0.999211	0.007243	0.999971
5	SVM	0.994479	0.994501	0.994479	0.994485	0.012243	0.999957
1	Decision Tree	0.998028	0.998037	0.998028	0.998027	0.071064	0.998098
3	ADABoost	0.662066	0.855655	0.662066	0.680578	0.645262	0.963105

Based on performance metrics (Accuracy, F1, Log-loss, and AUC), Logistic Regression and XGBoost provide the best overall classification performance, with perfectly calibrated predictions and clean probability separation. Models like Random Forest and GBM follow closely. AdaBoost underperforms significantly and may require further tuning or replacement.

🧠 Key Factors for Model Choice Factor Consideration Accuracy / F1 Logistic Regression, XGB, GBM all strong Log-loss (probability quality) Logistic Regression & XGB best Interpretability Logistic Regression > Tree-based models Scalability / Speed

Logistic Regression fast, XGB scales well Overfitting risk Check cross-validation — GBM/XGB may overfit Deployment constraints Any restrictions (e.g. explainability?)

🔍 Final Recommendation 🏆 Primary candidate for fine-tuning: XGBoost Performs nearly as well as logistic regression on all metrics

Offers better flexibility and non-linearity handling

Handles missing values, outliers, and feature interactions automatically

Highly tunable for performance, with great libraries/tools (e.g. Optuna, CV)

Hyperparameter Finetuning for XGBoost Model (XGB)

XGBoost Import Block

	home_Goals	away_Goals	home_GoalsHalfTime	home_xGoals	home_shots	home_ppda	h
9123	1	0	0	1.354740	10.000000	6.1111	
809	0	0	0	1.030380	19.000000	2.2500	
780	0	1	0	0.744526	15.000000	7.5758	
265	3	2	3	2.074880	11.000000	10.8571	
12190	1	1	0	1.754310	13.484856	3.3704	
...	...	...	...	...	...	...	
4626	1	4	1	1.495350	16.000000	10.6190	
11262	1	2	1	1.474980	13.000000	7.6000	
3731	1	3	1	0.600074	8.000000	10.0000	
439	2	0	1	0.569734	13.000000	10.9583	
1827	0	3	0	0.135570	3.000000	20.4286	

2536 rows × 20 columns

Random Search

```
dict_keys(['objective', 'use_label_encoder', 'base_score', 'booster', 'callbacks',
'colsample_bylevel', 'colsample_bynode', 'colsample_bytree', 'early_stopping_round
s', 'enable_categorical', 'eval_metric', 'gamma', 'gpu_id', 'grow_policy', 'import
ance_type', 'interaction_constraints', 'learning_rate', 'max_bin', 'max_cat_to_one
hot', 'max_delta_step', 'max_depth', 'max_leaves', 'min_child_weight', 'missing',
'monotone_constraints', 'n_estimators', 'n_jobs', 'num_parallel_tree', 'predicto
r', 'random_state', 'reg_alpha', 'reg_lambda', 'sampling_method', 'scale_pos_weigh
t', 'subsample', 'tree_method', 'validate_parameters', 'verbosity'])
```

```
Random Grid: {'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'colsample_bytr
ee': [0.8, 1.0], 'min_child_weight': [50], 'subsample': [0.8], 'learning_rate':
[0.01, 0.05, 0.1]}


Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

▸ **RandomizedSearchCV**

▸ **estimator: XGBClassifier**

▸ XGBClassifier

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7608 entries, 8859 to 11119
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   home_Goals              7608 non-null   int64
 1   away_Goals              7608 non-null   int64
 2   home_GoalsHalfTime      7608 non-null   int64
 3   home_xGoals             7608 non-null   float64
 4   home_shots              7608 non-null   float64
 5   home_ppda               7608 non-null   float64
 6   home_corners            7608 non-null   float64
 7   home_total_assists      7608 non-null   int64
 8   away_xGoals             7608 non-null   float64
 9   away_ppda               7608 non-null   float64
 10  away_total_assists      7608 non-null   int64
 11  away_total_red_cards    7608 non-null   int64
 12  home_shotsOnTarget_cat  7608 non-null   float64
 13  away_shotsOnTarget_cat  7608 non-null   float64
 14  home_total_assists_cat  7608 non-null   float64
 15  away_total_assists_cat  7608 non-null   float64
 16  home_Goals_cat          7608 non-null   float64
 17  away_Goals_cat          7608 non-null   float64
dtypes: float64(12), int64(6)
memory usage: 1.1 MB


Model Performance
Accuracy:         0.9992
Precision:        0.9992
Recall:           0.9992
F1-score:         0.9992
Mean Absolute Error: 0.0008

Model Performance
Accuracy:         0.9941
Precision:        0.9941
Recall:           0.9941
F1-score:         0.9941
Mean Absolute Error: 0.0063
Improvement of -0.51%
```

```
Model Performance
Accuracy:         0.9980
Precision:        0.9980
Recall:           0.9980
F1-score:         0.9980
Mean Absolute Error: 0.0020
Model Performance
Accuracy:         0.9933
Precision:        0.9933
Recall:           0.9933
F1-score:         0.9933
Mean Absolute Error: 0.0075
Improvement on dev: -0.47%


Fitting 3 folds for each of 4 candidates, totalling 12 fits
```

▸ **GridSearchCV**

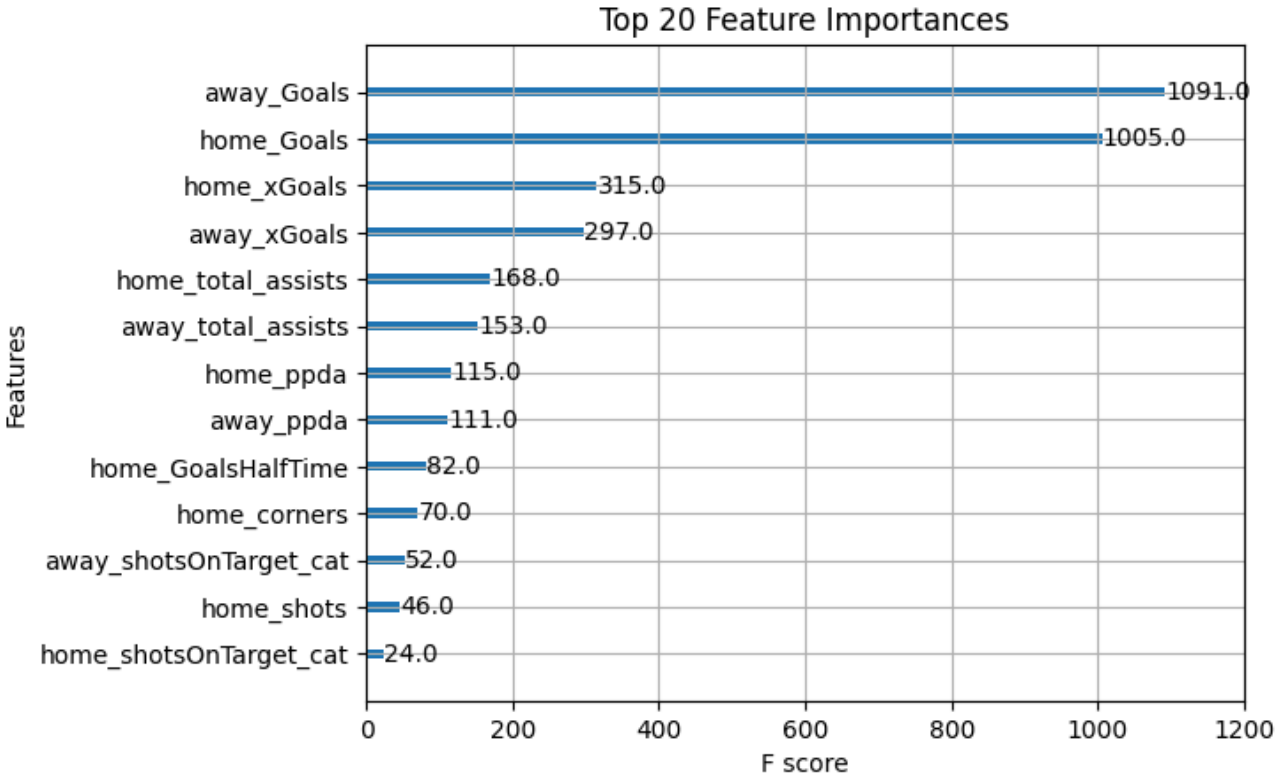
▸ **estimator: XGBClassifier**

▸ XGBClassifier

```
Model Performance
Accuracy:         0.9937
Precision:        0.9937
Recall:           0.9937
F1-score:         0.9937
Mean Absolute Error: 0.0079
Grid Search Improvement: -0.55%

Model Performance
Accuracy:         0.9937
Precision:        0.9937
Recall:           0.9937
F1-score:         0.9937
Mean Absolute Error: 0.0067
Grid Dev Improvement: -0.43%

Final model ready for production: XGBClassifier(base_score=0.5, booster='gbtree',
callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1.0,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric='logloss', gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.05, max_bin=256,
              max_cat_to_onehot=4, max_delta_step=0, max_depth=110,
              max_leaves=0, min_child_weight=50, missing=nan,
              monotone_constraints='()', n_estimators=400, n_jobs=0,
              num_parallel_tree=1, objective='multi:softprob', predictor='auto',
              random_state=0, reg_alpha=0, ...)
```




```
[CV] END colsample_bytree=0.8, max_depth=55, min_child_weight=45, n_estimators=100
0; total time=   9.8s
[CV] END colsample_bytree=0.8, max_depth=55, min_child_weight=50, n_estimators=100
0; total time=   9.9s
[CV] END colsample_bytree=0.8, max_depth=55, min_child_weight=50, n_estimators=100
0; total time=   9.7s
[CV] END colsample_bytree=0.8, max_depth=55, min_child_weight=50, n_estimators=100
0; total time=   9.9s
[CV] END colsample_bytree=0.8, max_depth=60, min_child_weight=45, n_estimators=100
0; total time=   9.5s
[CV] END colsample_bytree=0.8, max_depth=60, min_child_weight=45, n_estimators=100
0; total time=   7.2s
[CV] END colsample_bytree=0.8, max_depth=55, min_child_weight=45, n_estimators=100
0; total time=   9.6s
[CV] END colsample_bytree=0.8, max_depth=60, min_child_weight=50, n_estimators=100
0; total time=   7.3s
[CV] END colsample_bytree=0.8, max_depth=55, min_child_weight=45, n_estimators=100
0; total time=   9.6s
[CV] END colsample_bytree=0.8, max_depth=60, min_child_weight=50, n_estimators=100
0; total time=   7.3s
[CV] END colsample_bytree=0.8, max_depth=60, min_child_weight=45, n_estimators=100
0; total time=   9.7s
[CV] END colsample_bytree=0.8, max_depth=60, min_child_weight=50, n_estimators=100
0; total time=   7.5s
```